POLITECNICO DI TORINO
Repository ISTITUZIONALE

Memcomputing NP-complete problems in polynomial time using polynomial resources and collective states

Original
Memcomputing NP-complete problems in polynomial time using polynomial resources and collective states / Traversa, F. L.; Ramella, Chiara; Bonani, Fabrizio; Di Ventra, M.. - In: SCIENCE ADVANCES. - ISSN 2375-2548. - STAMPA. - 1:6(2015), pp. e1500031-e1500031. [10.1126/sciadv.1500031]

Availability:
This version is available at: 11583/2615026 since: 2015-07-16T09:42:25Z

Publisher:
American association for the advancement of science

Published
DOI:10.1126/sciadv.1500031

Terms of use:
openAccess

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

04 May 2023

COMPUTER SCIENCE

2015 © The Authors, some rights reserved; exclusive licensee American Association for the Advancement of Science. Distributed under a Creative Commons Attribution NonCommercial License 4.0 (CC BY-NC).
10.1126/sciadv.1500031

Memcomputing *NP*-complete problems in polynomial time using polynomial resources and collective states

Fabio Lorenzo Traversa,[1,2] Chiara Ramella,[2] Fabrizio Bonani,[2] Massimiliano Di Ventra[1]*

Memcomputing is a novel non-Turing paradigm of computation that uses interacting memory cells (memprocessors for short) to store and process information on the same physical platform. It was recently proven mathematically that memcomputing machines have the same computational power of nondeterministic Turing machines. Therefore, they can solve *NP*-complete problems in polynomial time and, using the appropriate architecture, with resources that only grow polynomially with the input size. The reason for this computational power stems from properties inspired by the brain and shared by any universal memcomputing machine, in particular intrinsic parallelism and information overhead, namely, the capability of compressing information in the collective state of the memprocessor network. We show an experimental demonstration of an actual memcomputing architecture that solves the *NP*-complete version of the subset sum problem in only one step and is composed of a number of memprocessors that scales linearly with the size of the problem. We have fabricated this architecture using standard microelectronic technology so that it can be easily realized in any laboratory setting. Although the particular machine presented here is eventually limited by noise—and will thus require error-correcting codes to scale to an arbitrary number of memprocessors—it represents the first proof of concept of a machine capable of working with the collective state of interacting memory cells, unlike the present-day single-state machines built using the von Neumann architecture.

INTRODUCTION

There are several classes of computational problems that require time and resources that grow exponentially with the input size when solved. This is true when these problems are solved with deterministic Turing machines, namely, machines based on the well-known Turing paradigm of computation, which is at the heart of any computer we use nowadays (*1*, *2*). Prototypical examples of these difficult problems are those belonging to the class that can be solved in polynomial (*P*) time if a hypothetical Turing machine—named nondeterministic Turing machine—could be built. They are classified as nondeterministic polynomial (*NP*) problems, and the machine is hypothetical because, unlike a deterministic Turing machine, it requires a fictitious "oracle" that chooses which path the machine needs to follow to get to an appropriate state (*1*, *3*, *4*). To date, it is not known whether *NP* problems can be solved in polynomial time by a deterministic Turing machine (*5*, *6*). If that were the case, we could finally provide an answer to the most outstanding question in computer science, namely, whether $NP = P$ (*1*).

Recently, a new paradigm, named "memcomputing" (*7*), has been advanced. It is based on the brain-like notion that one can process and store information within the same units (memprocessors) by means of their mutual interactions. This paradigm has its mathematical foundations on an ideal machine, alternative to the Turing one, that was formally introduced by two of the authors (F.T. and M.D.) and dubbed "universal memcomputing machine (UMM)" (*8*). It has been proven mathematically that UMMs have the same computational power of a nondeterministic Turing machine (*8*), but unlike the latter, UMMs are fully deterministic machines and, as such, they can actually be fabricated. A UMM owes its computational power to three main properties: intrinsic parallelism—interacting memory cells simultaneously and collectively change their states when performing computation; functional polymorphism—depending on the applied signals, the same interacting memory cells can calculate different functions; and finally, information overhead—a group of interacting memory cells can store a quantity of information that is not simply proportional to the number of memory cells itself.

These properties ultimately derive from a different type of architecture: the topology of memcomputing machines is defined by a network of interacting memory cells (memprocessors), and the dynamics of this network are described by a collective state that can be used to store and process information simultaneously. This collective state is reminiscent of the collective (entangled) state of many qubits in quantum computation, where the entangled state is used to solve efficiently certain types of problems such as factorization (*9*). Here, we prove experimentally that such collective states can also be implemented in classical systems by fabricating appropriate networks of memprocessors, thus creating either linear or nonlinear combinations out of the states of each memprocessor. The result is the first proof of concept of a machine able to solve an *NP*-complete problem in polynomial time using collective states.

The experimental realization of the memcomputing machine presented here, and theoretically proposed in (*8*), can solve the *NP*-complete (*10*) version of the subset sum problem (SSP) in polynomial time with polynomial resources. This problem is as follows: if we consider a finite set $G \in Z$ of cardinality n, is there a non-empty subset $K \in G$ whose sum is a given integer number s? As we discuss in the following paragraphs, the machine we built is analog and hence would be scalable to very large numbers of memprocessors only in the absence of noise or using some error-correcting codes. This problem derives from the fact that in the present realization, we use the frequencies of the collective state to encode information, and to maintain the energy of

[1]Department of Physics, University of California, San Diego, La Jolla, CA 92093, USA. [2]Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Turin, Italy.
*Corresponding author. E-mail: diventra@physics.ucsd.edu

the system bounded, the amplitudes of the frequencies are dampened exponentially with the number of memprocessors involved. However, this latter limitation is due to the particular choice of encoding the information in the collective state and could be overcome by using other realizations of digital memcomputing machines and using error-correcting codes. For example in (*8*), two of the authors (F.T. and M.D.) proposed a different way to encode a quadratic information overhead in a network of memristors that is not subject to this energy bound.

Another example in which information overhead does not need exponential growth of energy is, again, quantum computing. For instance, a close analysis of Shor's algorithm (*11*) shows that the collective state of the machine implements all at once (through the superposition of quantum states) an exponential number of states, each one with the same probability that decreases exponentially with the number of qubits involved. Successively, the quantum Fourier transform reorganizes the probabilities encoded in the collective state and "selects" those that actually solve the implemented problem (the prime factorization in the case of the Shor's algorithm). It is worth noticing, however, that quantum computing algorithms also necessarily require error-correcting codes for their practical implementation because of several unavoidable sources of noise (*9*).

Here, it is also worth stressing that our results do not answer the $NP = P$ question, because the latter has its solution only within the Turing machine paradigm: although a UMM is Turing-complete (*8*), it is not a Turing machine. In fact, (classical) Turing machines use states of single memory cells and do not use collective states. Finally, we mention that other unconventional approaches to the solution of *NP*-complete problems have been proposed (*6*, *12*–*16*); however, none of them reduces the computational complexity or keeps the physical resources from growing exponentially with the size of the problem. On the contrary, our machine can solve an *NP*-complete problem with only polynomial resources. As anticipated, this last claim is strictly valid for an arbitrary large input size only in the absence of noise.

RESULTS

Implementing the SSP

The machine we built to solve the SSP is a particular realization of a UMM based on the memcomputing architecture described in (*8*); namely, it is composed of a control unit, a network of memprocessors (computational memory), and a readout unit, as schematically depicted in Fig. 1. The control unit is composed of generators applied to each memprocessor. The memprocessor itself is an electronic module fabricated from standard electronic devices, as sketched in Fig. 2 and detailed in Materials and Methods. Finally, the readout unit is composed of a frequency shift module and two multimeters. All the components we have used employ commercial electronic devices.

The control unit feeds the memprocessor network with sinusoidal signals (that represent the input signal of the network) as in Fig. 1. It is simple to show that the collective state of the memprocessor network of this machine (which can be read at the final terminals of the network) is given by the real (up terminal) and imaginary (down terminal) part of the function

$$g(t) = 2^{-n}\prod_{j=1}^{n}\left(1+\exp[i\omega_j t]\right) \qquad (1)$$

where n is the number of memprocessors in the network and i the imaginary unit [see Materials and Methods or (*8*)]. If we indicate with $a_j \in G$ the jth element (integer with sign) of G, and we set the frequencies as $\omega_j = 2\pi a_j f_0$, with f_0 (fundamental frequency) equal for any memprocessor, we are actually encoding the elements of G into the memprocessors through the control unit feeding frequencies. Therefore, the frequency spectrum of the collective state (1) [or more precisely the spectrum of $g(t) - 2^{-n}$] will have the harmonic amplitude, associated with the normalized frequency $f = \omega/(2\pi f_0)$, proportional to the number of subsets $K \subseteq G$, whose sum s is equal to f . That is, if we read the spectrum of the collective state (1), the harmonic amplitudes are the solution of the SSP for any s. From this first analysis, we can make the following conclusions.

Information overhead. The memprocessor network is fed by n frequencies encoding the n elements of G, but the collective state (1) encodes all possible sums of subsets of G into its spectrum. It is well known (*5*) that the number of possible sums s (or equivalently the scaled frequencies f of the spectrum) can be estimated in the worst case as $O(A)$, where $A = \max[\Sigma_{a_j>0}\ a_j, -\Sigma_{a_j<0}\ a_j]$. Obviously A (sometimes called the capacity of the problem) has exponential growth (*17*) on the minimum number p of bits used to represent the elements of G [p is called precision of the problem, and we have $A = O(2^p)$, if we take the precision in bits]. Thus, the spectrum of the collective state (Eq. 1) encodes an information overhead that grows exponentially with the precision of the problem.

Computation time. The collective state (Eq. 1) is a periodic function of t with minimum period $T = 1/f_0$ because all frequencies involved in Eq. 1 are multiples of the fundamental frequency f_0. Therefore, T is the minimum time required to compute the solution of the SSP within the memprocessor network, and so it can be interpreted as the computation time of the machine. However, this computation time is independent of both n and p.

Energy expenditure. The energy required to compute the SSP can be estimated as that quantity proportional to the energy of the collective state in one period $E = \int_0^T |g(t)|^2 \mathrm{d}t$. By using Eq. 1, we have $E \leq \int_0^T \mathrm{d}t \leq 1/f_0$, so the energy needed for the computation is also independent of both n and p. It is worth remarking here that, to keep the energy bounded, all generators have the coefficient 0.5 (see Fig. 1) and then introduce (see Materials and Methods) the factor 2^{-n} in Eq. 1. This means that all frequencies involved in the collective state (Eq. 1) are dampened by the factor 2^{-n}. In the case of the ideal machine, that is, a noiseless machine, this would not represent an issue because no information would be lost. On the contrary, when noise is accounted for, the exponential factor represents the hardest limitation of the experimentally fabricated machine, which we reiterate is a technological limit for this particular realization of a memcomputing machine but not for all of them.

Reading the SSP solution

With this analysis, we have proven that the UMM represented in Fig. 1 can solve the SSP with n memprocessors, a control unit formed by $n + 1$ generators and taking a time T and an energy E independent of both n and p. Therefore, at first glance, it seems that this machine (without the readout unit) can solve the SSP using only resources polynomial (specifically, linear) in n. However, we need one more step: we have to read the result of the computation. Unfortunately, we cannot simply read the collective state (Eq. 1) using, for example, an oscilloscope and performing the Fourier conversion. This is because the most optimized algorithm to do this [see Materials and Methods and (*8*)] is exponential in p, that is, it has the same complexity of standard dynamic programming (*17*).

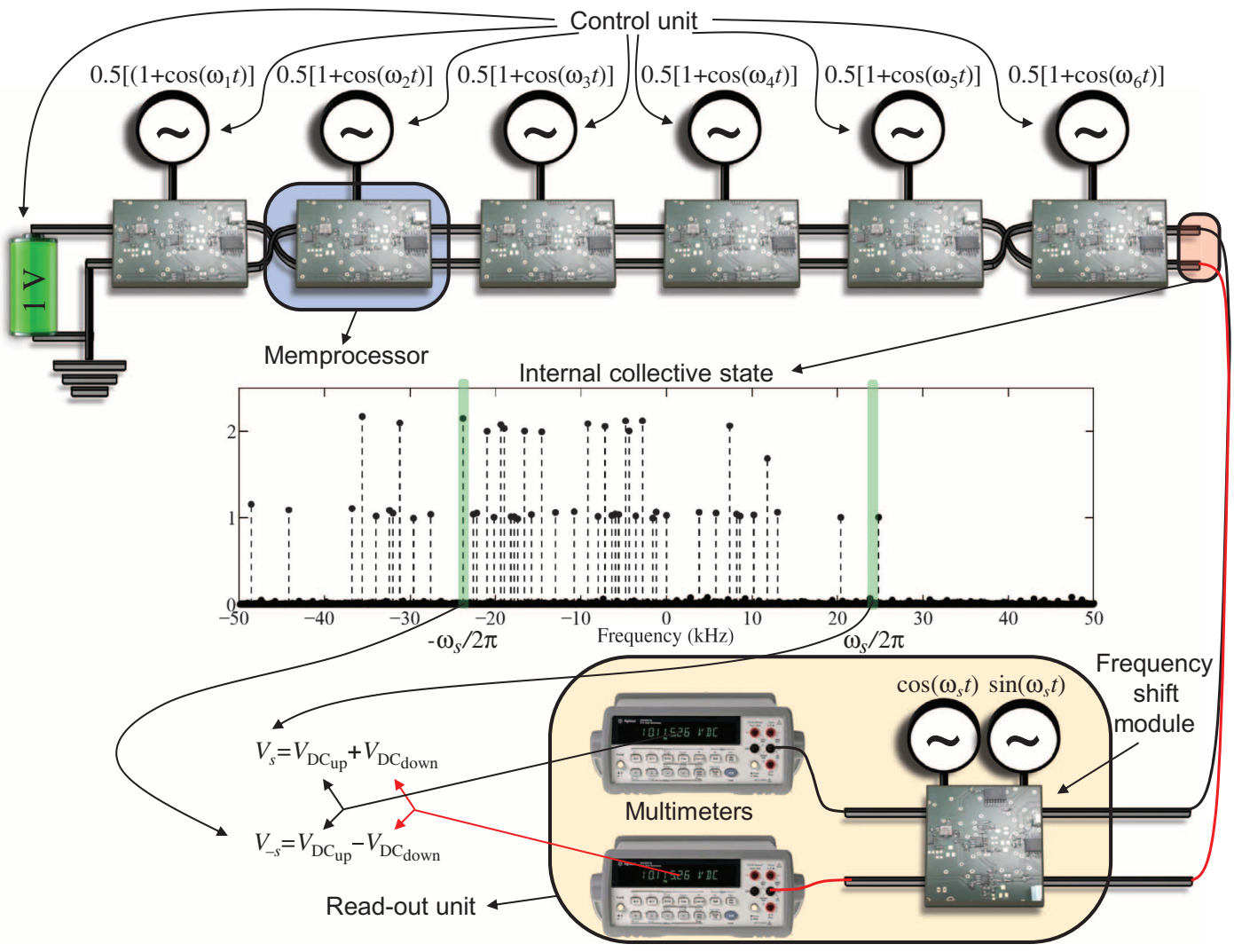

Fig. 1. Scheme of the memcomputing architecture used in this work to solve the subset sum problem. The central spectrum was obtained by the discrete Fourier transform of the experimental output of a network of six memprocessors encoding the set G = {130, −130, −146, −166, −44, 118} with fundamental frequency f_0 = 100 Hz.

However, a solution to this problem can be found by just using standard electronics to implement a readout unit capable of extracting the desired frequency amplitude without adding any computational burden. In Fig. 1, we sketch the readout unit we have used. It is composed of a frequency shift module and two multimeters. The frequency shift module is, in turn, composed of two voltage multipliers and two sinusoidal generators as depicted in Fig. 2, and it works as follows. If we connect to one multiplier the real part of a complex signal $v(t)$ and to the other multiplier the imaginary part, we obtain at the outputs $v_{up}(t)$ = Re[$v(t)$](exp[$i\omega_s t$] + exp[$-i\omega_s t$])/2 and $v_{down}(t)$ = − iIm[$v(t)$](exp[$i\omega_s t$]) − exp[$-i\omega_s t$]/2. The sum and difference of the two outputs are $v_{up} + v_{down}$ = Re[$v(t)$exp[$-i\omega_s t$]] and $v_{up} - v_{down}$ = Re[$v(t)$exp[$i\omega_s t$]]. From basic Fourier calculus, $v(t)$exp[$-i\omega_s t$] and $v(t)$exp[$i\omega_s t$] are the frequency spectrum shifts of $\omega_s/(2\pi)$ and $-\omega_s/(2\pi)$ for the function $v(t)$, respectively. Consequently, if we read the DC voltages $V_{DC_{up}}$ and $V_{DC_{down}}$ of $v_{up}(t)$ and $v_{down}(t)$ using two multimeters and perform the sum $V_{DC_{up}} + V_{DC_{down}}$ and difference $V_{DC_{up}} - V_{DC_{down}}$, we obtain the amplitudes V_s and V_{-s} of the harmonics at the frequencies $\omega_s/(2\pi)$ and $-\omega_s/(2\pi)$, respectively (see Fig. 1).

Hence, by feeding the frequency shift module with the Re[$g(t)$] and Im[$g(t)$] from Eq. 1, reading the output with the two multimeters, and performing the sum and difference of the final outputs, we obtain the harmonic amplitude for a particular normalized frequency f according to the external frequency ω_s of the frequency shift module. In other words, without adding any additional computational burden and time, we can solve the SSP for a given s by properly setting the external frequency of the frequency shift module.

Notably, if we wanted to simulate our machine by including the readout unit, the computational complexity would be $O(2^p)$ [close to the standard dynamic programming that is $O(n2^p)$ (*17*)]. From the Nyquist-Shannon sampling theorem (*18*), the minimum number of samples of the shifted collective state (that is, the outputs of the frequency shift module) must be equal to the number of frequencies of the signal [in our case of $O(2^p)$] to accurately evaluate even one of

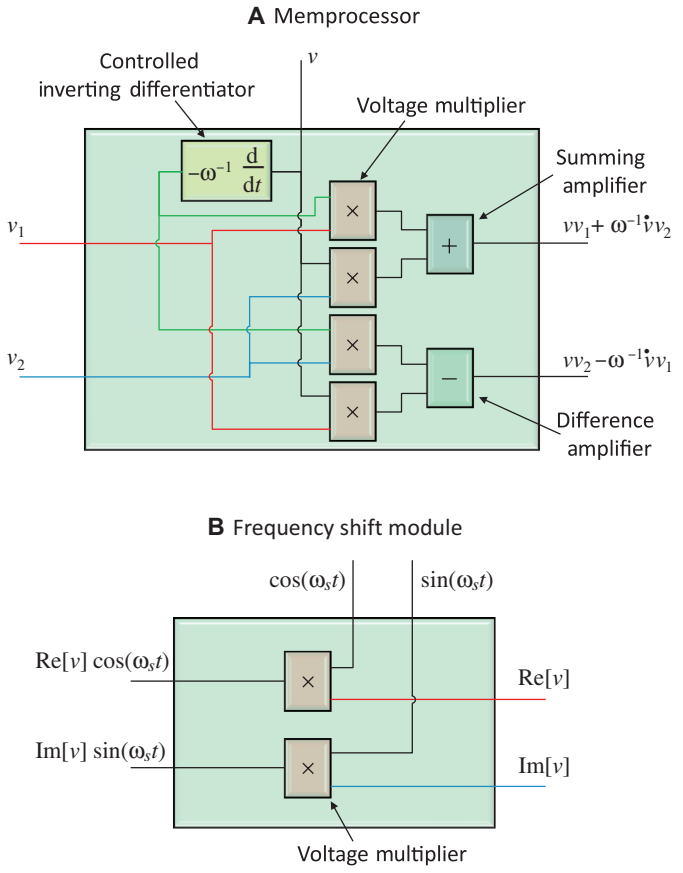

Fig. 2. Schematic of the modules. (**A**) Simplified schematic of the memprocessor architecture used in this work to solve the SSP (more details can be found in Materials and Methods). (**B**) Schematic of the frequency shift module.

the harmonic amplitudes (*19*). The last claim can be intuitively seen from this consideration: the DC voltage $V_{DC_{up}}$ must be calculated in the simulation by evaluating the integral $V_{DC_{up}} = T^{-1}\int_0^T v_{up}(t)dt$, and this requires at least $O(2^p)$ samples for an accurate evaluation (*18*). On the other hand, the multimeter of the hardware implementation, being essentially a narrow low-pass filter, performs an analog implementation of the integral over a continuous time interval T (independent of n and p), directly providing the result and thus avoiding the need to sample the waveform and compute the integral.

In Fig. 3, the absolute value of the spectrum of the collective state for networks of four, five, and six memprocessors is compared with the theoretical results given by the spectrum of Eq. 1 (see Materials and Methods for more details on the hardware and measurement process). Non-idealities of the circuit and electronic noise in general are the sources of the small discrepancies with respect to the theoretical results. Nevertheless, the machine we fabricated demonstrates that using the collective state of all memprocessors, instead of the uncoupled states of the individual memory units, we can carry out difficult computing tasks (*NP*-complete problems) with polynomial resources. Finally, in Table 1, the measurements at the readout circuit are listed for different harmonics for a six-memprocessor network. The precision is up to the third digit as can be seen from the comparison with the theoretical results.

DISCUSSION

Scalability and error correction

As anticipated, the machine we have built would ultimately be limited by unavoidable noise sources, thus requiring error-correcting codes. However, we prove here that under the assumption of low noise, additive white noise does not affect the machine output. Therefore, only non-idealities of the devices and colored noise represent a limit for this particular machine. To analyze this issue, we first consider the simplest case of independent white Gaussian additive noise sources for each memprocessor. Under this hypothesis, each memprocessor has complex noisy input of the form $v_{j_{in}} = 2^{-1}(1+\exp[i\omega_j t]) + \varepsilon_j(t)$. When we connect two memprocessors, the output at the terminals of the second memprocessor is given by

$$v_{2_{out}} = v_{1_{in}} v_{2_{in}} = 2^{-2}(1+e^{i\omega_1 t})(1+e^{i\omega_2 t}) + 2^{-1}(1+e^{i\omega_1 t})\varepsilon_2(t) + 2^{-1}(1+e^{i\omega_2 t})\varepsilon_1(t) + \varepsilon_1(t)\varepsilon_2(t) \quad (2)$$

If the noise term is small enough (low noise components), the quantity $\varepsilon_1(t)\varepsilon_2(t)$ is negligible compared to the other terms and can be neglected. Following the same steps, the network composed of n memprocessors has collective state (under low noise conditions) of the form

$$g(t) = v_{n_{out}} = 2^{-n}\prod_{j=1}^{n}(1+e^{i\omega_j t}) + 2^{-n+1}\sum_{k=1}^{n}\varepsilon_k(t)\prod_{\substack{j=1\\ j\neq k}}^{n}(1+e^{i\omega_j t}) = g_S(t) + g_N(t) \quad (3)$$

Using Eq. 3, we can calculate the signal-to-noise ratio as the ratio between the power of the signal and the power of the noise. The signal power (neglecting noise) is simply given by $S = E = \int_0^T |g_S(t)|^2 dt \approx T$, and the noise power is given by $N = \int_0^T E[|g_N(t)|^2]dt$, where $E[\cdot]$ is the expectation value operator. Because the noise sources are independent, we have $E[|g_N(t)|^2] \approx \Sigma_{k=1}^n E[|\varepsilon_k(t)|^2]$, and, since the sources are white, $E[|g_N(t)|^2]$ is independent of time. Finally, assuming $E[|\varepsilon_k(t)|^2] = E[|\varepsilon_j(t)|^2]$ for any j and k, we have $E[|g_N(t)|^2] \approx TnE[|\varepsilon|^2]$, and the signal-to-noise ratio reads

$$\frac{S}{N} \approx \frac{1}{nE[|\varepsilon|^2]}. \quad (4)$$

However, the noise being white, the noise power spectrum, according to Eq. 3, is distributed among the different harmonics with weights given by the coefficients in Eq. 3 that are exponentially decreasing with n. Therefore, the signal-to-noise ratio for each harmonic has the same order of magnitude of Eq. 4, and hence, the white noise (under conditions of low noise) does not affect the machine when we scale it up.

On the other hand, non-idealities, colored and $1\backslash f$ noise and other non-Gaussian noise sources, have spectra that are not distributed on the harmonics as white noise. For example, the $1\backslash f$ noise accumulates on the output harmonic during the process of measurement. Nevertheless, if the total noise can be considered low, Eq. 3 is still valid but the ratio (Eq. 4) should be computed on a frequency by frequency basis. In this case, we can apply Shannon's noisy-channel coding theorem (*20*) with our machine interpreted as a noisy channel with capacity C. The input

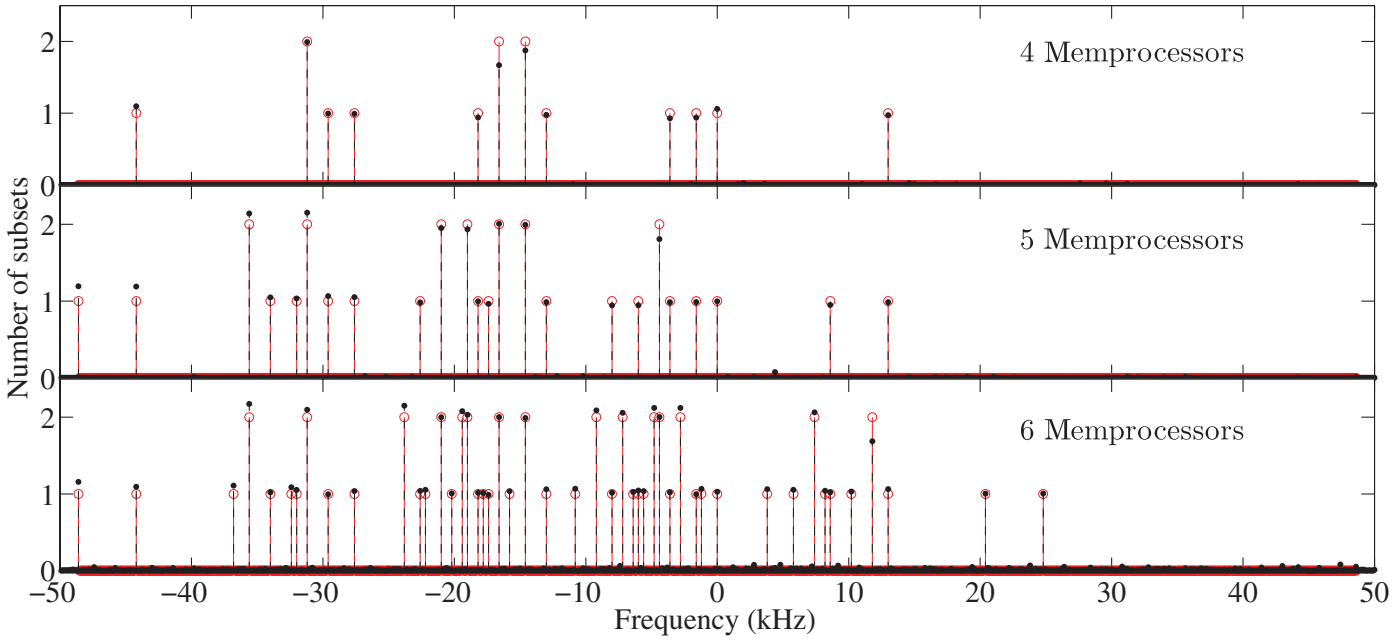

Fig. 3. Spectra of the internal collective state of three different networks with fundamental frequency f_0 = 100 Hz. The four-memprocessor network encodes the set G = {130, −130, −146, −166}. The five-memprocessor network encodes G = {130, −130, −146, −166, −44}. The six-memprocessor network encodes G = {130, −130, −146, −166, −44, 118}.

Table 1. Measurements from the readout unit of Fig. 1 for a six-memprocessor network with fundamental frequency f_0 = 100 Hz encoding the set G = {130, −130, −146, −166, −44, 118}. In the fifth and sixth columns, the voltages are respectively given by $V_s = 2^6(V_{DC_{up}} + V_{DC_{down}})$ and $V_{-s} = 2^6(V_{DC_{up}} + V_{DC_{down}})$. The last two columns are the analytical results.

$\|S\|$	$\omega_{s/(2\pi)}$ (kHz)	$V_{DC_{up}}$ (mV)	$V_{DC_{down}}$ (mV)	V_s (V)	V_{-s} (V)	No. of subset sum s	No. of subset $-s$
0	0	31.7	0	1.02	1.02	1	1
74	7.4	15.3	15.0	1.94	0.02	2	0
130	13.0	−0.2	14.9	0.94	−0.97	1	1
146	14.6	14.8	15.8	−0.06	1.96	0	2
248	24.8	7.6	7.2	0.95	0.02	1	0
485	48.5	−0.4	−0.7	−0.07	0.02	0	0
486	48.6	−8.9	6.6	−0.14	0.99	0	1

is given by the control unit, and the output is the collective state $g(t)$. The Shannon theorem then states that for any $\varepsilon > 0$ and $R < C$, there exists a code of length N and rate $\geq R$ and a decoding algorithm, such that the maximal probability of block error is $\leq\varepsilon$.

In addition, using the Shannon-Hartley theorem (*21*) we have for frequency-dependent noise, the capacity C can be calculated as

$$C = \int_0^B \log_2\left(1 + \frac{S(f)}{N(f)}\right) \mathrm{d}f, \tag{5}$$

where B is the bandwidth of the channel. In our case, the lower bound of B can be taken as linear in the number of memprocessors, that is, $B = B_0 n$, with B_0 a constant, and S/N is given by Eq. 4. We finally have (for large n)

$$C \approx \int_0^B \log_2\left(1 + \frac{1}{nE[\varepsilon^2(f)]}\right) df \approx \frac{B_0}{E[\varepsilon^2]\ln 2} \quad \text{where } \overline{E[\varepsilon^2]}^{-1} = \lim_{n\to\infty}\int_0^{nB_0} E[\varepsilon^2(f)]^{-1} df \tag{6}$$

We therefore conclude that our machine compresses data in an exponential way with constant capacity. At the output, we have an exponentially decreasing probability of finding one solution of the SSP when we implement the algorithm by brute force, that is, without any error-correcting coding. However, from the Shannon theorem, there exists a code that allows us to send the required information with bounded error. The question then is whether there is a polynomial code that accomplishes this task. We briefly discuss the question here heuristically. If our machine were Turing-like, then the polynomial code could exist only if $NP = P$. Instead, our machine is not Turing-like, and the channel can perform an exponential number of operations at the same time. Because this is similar to what quantum computing does when solving difficult problems such as factorization, and we know that for quantum computing polynomial correcting codes do exist (*9*), we expect that similar coding can be applied to our specific machine as well.

CONCLUSION

In summary, we have demonstrated experimentally a deterministic memcomputing machine that is able to solve an NP-complete problem in polynomial time (actually in one step) using only polynomial resources. The actual machine we built clearly suffers from technological limitations, that impair its scalability due to unavoidable noise. These limitations derive from the fact that we encode the information directly

into frequencies, and so ultimately into energy. This issue could, however, be overcome either using error correcting codes or with other UMMs that use other ways to encode such information and are digital at least in their input and output. Irrespective, this machine represents the first experimental realization of a UMM that uses the collective state of the whole memprocessor network to exploit the information overhead theoretically introduced in (*8*). Finally, it is worth mentioning that the machine we have fabricated is not a general purpose one. However, other realizations of UMMs are general purpose and can be easily built with available technology (*22*–*26*). Their practical realization would thus be a powerful alternative to current Turing-like machines.

MATERIALS AND METHODS

Experimental design

The operating frequency range of the experimental setup needs to be discussed with the measurement target in mind. For the measurement of the entire collective state, the limiting frequency is due to the oscilloscope we use to sample the full signal at the output of the memprocessor network. On the other hand, the measurement of some isolated harmonic amplitudes using the readout unit transfers this bottleneck to the voltage generators and internal memprocessor components. It is worth stressing that measuring the collective state is not the actual target of our work because it has the same complexity of the standard algorithms for the SSP as discussed in Results and in (*8*). Here, for completeness, we provide measurements of the collective state only to prove that the setup works properly. The actual frequency range of the setup is discussed in the next section.

Setup frequency range

Let us consider $a_j \in G$ and the integer

$$A = \max\left\{-\sum_{a_j<0} a_j, \sum_{a_j>0} a_j\right\}. \quad (7)$$

We also consider $f_0 \in \mathbb{R}$ and we encode the a_j in the frequencies by setting the generators at frequencies $f_j = |a_j|f_0$ so the maximum frequency of the collective state is $f_{\max} = Af_0$. From these considerations, we can first determine the range of the voltage generators: it must allow for the minimum frequency (resolution)

$$f_{g_{\min}} \leq f_0 \quad (8)$$

and maximum frequency (bandwidth)

$$f_{g_{\max}} \geq f_0 \max_G\{|a_j|\}. \quad (9)$$

We used Agilent 33220A Waveform Generator (*27*), which has 1-μHz resolution and 20-MHz bandwidth. This means that, in principle, we can accurately encode G when composed of integers with a precision up to 13 digits (which is the same precision as the standard double-precision integers) provided that a stable and accurate external clock reference, such as a rubidium frequency standard, is used. This is because the internal reference of such generators introduces a relative uncertainty on the synthesized frequency in the range of some parts-per-billion (10^{-9}), thus limiting the resolution at high frequency, down to a few millihertz at the maximum frequency. However, as anticipated, this issue can be resolved by using an external reference providing higher accuracies. On the other hand, note that the frequency range can be, in principle, increased by using wider bandwidth generators, up to the gigahertz range.

Another frequency limitation concerning the maximum operating frequency is given by the electronic components of the memprocessors. In fact, the active elements necessary to implement the memprocessor modules in hardware have specific operating frequencies that cannot be exceeded. Discrete operational amplifiers (OP-AMPs) are the best candidates for this implementation because of their flexibility in realizing different types of operations (amplification, sum, difference, multiplication, derivative, etc.). Their maximum operating frequency is related to the gain-bandwidth product (GBWP). We used standard high-frequency OP-AMP that can reach GBWP up to a few gigahertz. However, such amplifiers usually show high sensitivity to parasitic capacitances and stability issues (for example, a limited stable gain range). The typical maximum bandwidth of such OP-AMPs that ensures unity gain stability and acceptable insensitivity to parasitics is of the order of a few tens of megahertz, thus compatible with the bandwidth of the Agilent 33220A Waveform Generator. Therefore, we can set quantitatively the last frequency limit related to the hardware as

$$f_{\text{OP-AMP}_{\max}} > Af_0, \quad (10)$$

and ensure optimal OP-AMP functionality. Finally, using Eqs. 8 to 10, we can find a reasonable f_0 satisfying the frequency constraints.

Memprocessor

The memprocessor, synthetically discussed in Results and sketched in Fig. 2, is shown in fig. S1A, and a more detailed circuit schematics is given in fig. S1B. Each module has been realized as a single printed circuit board (PCB), and connections are performed through coaxial cables with BNC terminations. According to Fig. 2, each memprocessor must perform one derivative $-\omega_j^{-1}\frac{d}{dt}$, four multiplications, one sum, and one difference. Because $v(t) = 0.5[1 + \cos(2\pi f_0 a_j\, t)]$ and $\omega_j = 2\pi f_0 a_j$, then $-\omega_j^{-1}\dot{v}(t) = 0.5\sin(2\pi f_0 a_j t)$, that is, the quadrature signal with respect to the input $v(t)$. This can be easily obtained with the simple OP-AMP–based inverting differentiator depicted in fig. S1B, designed to have unitary gain at frequency ω_j . Similarly, an inverting summing amplifier and a difference amplifier can be realized as sketched in fig. S1B to perform the sum and difference of voltage signals, respectively. The OP-AMP selected is the Texas Instruments LM7171 Voltage Feedback Amplifier.

Implementing multiplication is slightly more challenging. OP-AMP–based analog multipliers are very sensitive circuits. Therefore, they need to be carefully calibrated. This makes a discrete OP-AMP–based realization challenging and the integration expensive. We thus adopted a preassembled analog multiplier: the Texas Instrument (Burr-Brown) MPY634 Analog Multiplier, which ensures four-quadrant operation, good accuracy, and wide-enough bandwidth (10 MHz). The only drawback of this multiplier is that the maximum precision is achieved with a gain of 0.1. Therefore, because the input signals in general are small, this further lowering of the precision can make the output signal comparable to the offset voltages of the subsequent OP-AMP stages. For this reason, we have included in the PCB a gain stage (inverting amplifier) before each output to compensate for the previous signal inversion and lowering. These stages also permit manual offset adjustment by means of a tunable network added to the non-inverting input, as shown

in the schematic. Finally, a low-pass filter with corner frequency $f_c \gg Af_0$ has been added to the outputs to limit noise. Figure S1A shows a picture of one of the modules, which have been realized on a 100 × 80–mm PCB. The power consumption of each module is high because all of the 10 active components work with ±15 V supply. OPAMPs have a quiescent current of 6.5 mA, whereas the multipliers have a quiescent current of 4 mA, yielding a total DC current of about 50 mA per module.

Finally, we briefly discuss how connected memprocessors work. From Fig. 2, if we have $v(t) = 0.5[1 + \cos(\omega t)]$, $v_1 = \mathrm{Re}[f(t)]$ and $v_2 = \mathrm{Im}[f(t)]$ for an arbitrary complex function $f(t)$, at the output of the memprocessor, we will find

$$v_{1_{out}} = 0.5[1 + \cos(\omega t)]\mathrm{Re}[f(t)] - 0.5\sin(\omega t)\mathrm{Im}[f(t)] = \mathrm{Re}[0.5(1 + e^{i\omega t})f(t)] \quad (11)$$

$$v_{2_{out}} = 0.5[1 + \cos(\omega t)]\mathrm{Im}[f(t)] + 0.5\sin(\omega t)\mathrm{Re}[f(t)] = \mathrm{Im}[0.5(1 + e^{i\omega t})f(t)] \quad (12)$$

Because we can only set positive frequencies for the generators, to encode negative frequencies (that is, a negative a_j) we can simply invert the input and output terminals as depicted in fig. S1C. Therefore, if we set $f(t) = 1$ for the first memprocessor, that is, $v_1 = 1$ and $v_2 = 0$ at the output of the first memprocessor, we will find the real and imaginary parts of $0.5(1 + \exp[i\omega_1 t])$ that will be the new $f(t)$ for the second memprocessor. Proceeding in this way, we find the collective state (1) at the end of the last memprocessor.

Analysis of the collective state

To test if the memprocessor network correctly works, we carried out the measurement of the full collective state $g(t)$ at the end of the last memprocessor. This task requires an extra discussion on the operating frequency range. Indeed, the instrument we used to acquire the output waveform is the LeCroy WaveRunner 6030 Oscilloscope. The measurement process consists of acquiring the output waveforms and applying the fast Fourier transform in software. The collective state being a purely periodic signal, that is, a signal containing only frequency multiples of f_0 and with known maximum frequency $f_{max} = Af_0$, from the discrete Fourier transform theory and the Nyquist-Shannon sampling theorem (*8*, *18*), we need to sample the interval $[0, 1/f_0]$ into $N = 2f_{max}/f_0 + 1$ subinterval of width $\Delta t = (Nf_0)^{-1}$ to compute the *exact* spectrum of $g(t)$. That is, we need samples $g(t_j)$ with $t_j = k\Delta t$ and $k = 0, \ldots, N - 1$ to compute the exact spectrum of $g(t)$. Therefore, with the oscilloscope, we must be able to acquire at least $N + 1$ samples of $g(t)$ into the time interval $[0, 1/f_0]$.

This relationship turns out to be a constraint on the usable frequency range because we must perform the measurement in a reasonable time and we cannot exceed the maximum sampling frequency of the instrument that we use for acquisition or its maximum memory capability. In our experimental proof, the LeCroy WaveRunner 6030 Oscilloscope is characterized by 350 MHz bandwidth, 2.5 GSa/s sampling rate, and 105 discretization points when saving waveforms in ascii format (that is, $N_{Omax} = 10^5$). The bandwidth of the oscilloscope is very large; thus, it is not really a constraint, whereas the limit N_{Omax} is. We have the constraint

$$f_{max} \leq \frac{f_0}{2}(N_{O_{max}} - 1). \quad (13)$$

With this value, choosing $f_0 = 1$ Hz allows us to have $A \leq (10^5 - 1)/2$, $f_{max} \lesssim 50$ kHz and requires 1-s measurement time, which is a long time in electronics. Therefore, without varying A, we can choose a larger f_0 that allows for a smaller measurement time. We choose $f0 = 100$ Hz, which means a measurement time of only a few tens of milliseconds, and $f_{max} \lesssim 5$ MHz.

Experimental setup

The laboratory setup we used is sketched in fig. S2A, whereas fig. S3 shows a picture of the same. The order of cascade connection of the memprocessors is arbitrary. For this test, we ordered the module such that $G = \{130, -130, -146, -166, -44, 118\}$ (see fig. S2A) to have the two memprocessors related to the two positive numbers (130 and 118) at the beginning and end of the chain, respectively, thus minimizing the number of "swapped" connections (see fig. S1C). A two-output power supply (model Agilent E3631A) is used to generate both the 0 and 1 V at the inputs of the first memprocessor and the ±15 V supply for all the modules (parallel connection). The input $v(t)$ of each module is generated by the Agilent 33220A waveform generator, whereas the output is observed through both an oscilloscope (model LeCroy WaveRunner 6030; see fig. S2C) and a multimeter (model Agilent 34401A). In particular, the oscilloscope is used for the AC waveform, whereas the multimeter measured the DC component to avoid errors due to the oscilloscope probes, which are very inaccurate at DC and may show DC offsets up to tens of millivolts.

Another issue concerns the synchronization of the generators. The six generators we used must share the same time base, and they must have the same starting instant (the $t = 0$ instant) when all the cosine waveforms must have an amplitude of 0.5 V. To have a common time base, we used the 10-MHz time base signal of one of the generators (master), and we connected the master output signal to all the other (slave) generators at the 10-MHz input. In this way, they ignore their own internal time base and lock to the external one. To have a common $t = 0$ instant, we must run the generators in the infinite burst mode. In this mode, the generators produce no output signal until a trigger input is given, and then they run indefinitely until they are manually stopped. The trigger input can be external or manual (soft key): the master device is set up to expect a manual input, whereas the slave devices are controlled by an external trigger coming from the master. Finally, to correctly visualize the output waveforms, we must also "synchronize" the oscilloscope. The trigger signal of the oscilloscope must have the same frequency of the signal to be plotted, or at least one of its subharmonics. Otherwise, we see the waveform moving on the display, and in case two or more signals are acquired, we lose the information concerning their phase relation. To solve this problem, we connect the external trigger input of the oscilloscope to a dedicated signal generator, producing a square waveform at frequency f_0, which is the greatest common divisor of all the possible frequency components of the output signals. This dedicated generator is also used as the master for synchronization. Figure S2B shows how the generators must be connected to obtain the required synchronization.

SUPPLEMENTARY MATERIALS

Supplementary material for this article is available at http://advances.sciencemag.org/cgi/content/full/1/6/e1500031/DC1

Fig. S1. Memprocessor module.

Fig. S2. Schematic and picture of the test bench for the laboratory experiment.
Fig. S3. Pictures of the laboratory test bench.

REFERENCES AND NOTES

1. M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman & Co., New York, 1990).
2. J. L. Hennessy, D. A. Patterson, *Computer Architecture: A Quantitative Approach* (Morgan Kaufmann Publishers Inc., San Francisco, CA, ed. 4, 2006).
3. A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* **42**, 230 (1936).
4. A. M. Turing, *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life, Plus The Secrets of Enigma* (Oxford Univ. Press, New York, 2004).
5. S. Arora, B. Barak, *Computational Complexity: A Modern Approach* (Cambridge Univ. Press, Cambridge, 2009).
6. K. Wu, J. G. de Abajo, C. Soci, P. P. Shum, N. I. Zheludev, An optical fiber network oracle for NP-complete problems. *Light Sci. Appl.* **3**, e147 (2014).
7. M. Di Ventra, Y. V. Pershin, The parallel approach. *Nat. Phys.* **9**, 200–202 (2013).
8. F. L. Traversa, M. Di Ventra, *(preprint on arXiv:1405.0931) IEEE Transaction on Neural Networks and Learning Systems, DOI: 10.1109/TNNLS.2015.2391182* (2015).
9. M. A. Nielsen, I. L. Chuang, *Quantum Computation and Quantum Information (Cambridge Series on Information and the Natural Sciences)* (Cambridge Univ. Press, Cambridge, ed. 10, 2010).
10. R. M. Karp, Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958–2008* (Springer, Berlin/Heidelberg, 2010), pp. 219–241.
11. P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**, 1484–1509 (1997).
12. D. Woods, T. J. Naughton, Optical computing: Photonic neural networks. *Nat. Phys.* **8**, 257 (2012).
13. T. D. Kieu, Quantum algorithm for Hilbert's tenth problem. *Int. J. Theor. Phys.* **42**, 1461–1478 (2003).
14. L. M. Adleman, Molecular computation of solutions to combinatorial problems. *Science* **266**, 1021–1024 (1994).
15. Z. Ezziane, DNA computing: Applications and challenges. *Nanotechnology* **17**, R27 (2006).
16. M. Oltean, Solving the Hamiltonian path problem with a light-based computer. *Nat. Comput.* **7**, 57–70 (2008).
17. S. Dasgupta, C. Papadimitriou, U. Vazirani, *Algorithms* (McGraw-Hill, New York, 2008).
18. F. Bonani, F. Cappelluti, S. D. Guerrieri, F. L. Traversa, Harmonic balance simulation and analysis. In *Wiley Encyclopedia of Electrical and Electronics Engineering* (Wiley, New York, 2014).
19. G. Goertzel, An algorithm for the evaluation of finite trigonometric series. *Am. Math. Mon.* **65**, 34–35 (1958).
20. C. E. Shannon, A mathematical theory of communication. *Bell Syst. Tech. J.*, **27**, 379 (1948).
21. H. Taub, D. L. Schilling, *Principles of Communication Systems* (McGraw-Hill Higher Education, New York, ed. 2, 1986).
22. F. L. Traversa, F. Bonani, Y. V. Pershin, M. Di Ventra, Dynamic computing random access memory. *Nanotechnology* **25**, 285201 (2014).
23. R. Waser, M. Aono, Nanoionics-based resistive switching memories. *Nat. Mater.* **6**, 833–840 (2007).
24. D. B. Strukov, G. S. Snider, D. R. Stewart, R. S. Williams, The missing memristor found. *Nature* **453**, 80–83 (2008).
25. T. Driscoll, H. T. Kim, B. G. Chae, B. J. Kim, Y. W. Lee, N. M. Jokerst, S. Palit, D. R. Smith, M. Di Ventra, D. N. Basov, Memory metamaterials. *Science* **325**, 1518–1521 (2009).
26. A. Chanthbouala, V. Garcia, R. O. Cherifi, K. Bouzehouane, S. Fusil, X. Moya, S. Xavier, H. Yamada, C. Deranlot, N. D. Mathur, M. Bibes, A. Barthélémy, J. Grollier, A ferroelectric memristor. *Nat. Mater.* **11**, 860 (2012).
27. Note that when we use digital voltage generators, we need to control the frequency with precision proportional to *p* Therefore, generating waveforms with digital generators should require exponential resources in p because of the time sampling. However, this can be, in principle, overcome using synchronized analog generators designed for this specific task.

Acknowledgments: The hardware realization of the machine presented here was supported by Politecnico di Torino through the Neural Engineering and Computation Lab initiative. M.D.V. acknowledges partial support from Center for Magnetic Recording Research. This work was also partly supported by EURAMET and by the European Union under an EMRP Research Grant. **Author contributions:** F.L.T., C.R., F.B., and M.D.V. designed the experiment and analyzed the data. C.R. assembled the circuits and performed the measurements. F.L.T., C.R., F.B., and M.D.V. co-wrote the paper. **Data and materials availability:** Data can be requested directly from C.R. (chiara.ramella@polito.it) or F.L.T. (ftraversa@physics.ucsd.edu). **Competing interests:** The authors declare that they have no competing interests.

Submitted 9 January 2015
Accepted 6 May 2015
Published 3 July 2015
10.1126/sciadv.1500031

Citation: F. L. Traversa, C. Ramella, F. Bonani, M. Di Ventra, Memcomputing *NP*-complete problems in polynomial time using polynomial resources and collective states. *Sci. Adv.* **1**, e1500031 (2015).